Homework 1: DDPM Implementation

CMU 10-799: Diffusion & Flow Matching

Spring 2026

Due: Saturday, January 24, 2026 at 11:59 PM ET **Total Points:** 100

Late Due Date: Monday, January 26, 2026 at 11:59 PM ET

Submission: Gradescope https://www.gradescope.com/courses/1207241

Starter Code: https://github.com/KellyYutongHe/cmu-10799-diffusion/

Introduction

Welcome to your first homework in CMU 10799 Diffusion & Flow Matching!

In this homework, you'll implement a Denoising Diffusion Probabilistic Model (DDPM) [Ho et al., 2020] from scratch and train it to generate realistic face images. But more than just "getting it to work," we want you to develop the intuition and debugging skills that researchers use every day. The structure of this homework mirrors a pseudo research workflow:

1. Understand your data: Before writing any model code, explore what you're trying to generate
2. Build intuition: Use the 1D playground to see diffusion in action (optional but recommended)
3. Implement: Write the core DDPM algorithm and U-Net architecture
4. Debug: Learn to diagnose common failure modes
5. Experiment: Run ablations to understand design choices
6. Reflect: Document what you learned and what you'd try next

This homework is AI-friendly. You may use any AI coding assistants, chatbots, or reference implementations. You may also use any other resources that you can find on the Internet. At the end of the homework, you'll document what resources you used.

Part 0: Setup your codebase (0 point)

First let's get started and setup your environments and codebase by forking and cloning the startup code repository: https://github.com/KellyYutongHe/cmu-10799-diffusion.

```
git clone https://github.com/KellyYutongHe/cmu-10799-diffusion.git
cd cmu-10799-diffusion

# Run setup script (auto-detects your hardware)
./setup-uv.sh     # Recommended (faster)

# Activate environment
source .venv-*/bin/activate
```

What You'll Implement:

File	What to implement
`src/data/celeba.py`	Data preprocessing transforms
`src/methods/ddpm.py`	Full DDPM algorithm
`src/models/unet.py`	U-Net architecture using provided blocks
`configs/`	Create your own model configs
`train.py`	Add sampling for logging
`sample.py`	Add your sampling scheme

What We Provide:

- **U-Net [Ronneberger et al., 2015] building blocks** (`blocks.py`): ResBlock, AttentionBlock, SinusoidalPositionEmbedding, Downsample, Upsample, etc
- **Training infrastructure**: Mixed precision, gradient clipping, checkpointing, logging
- **EMA**: Exponential moving average for stable sampling
- **Evaluation**: KID computation via torch-fidelity [Obukhov et al., 2020]
- **Modal integration**: Cloud GPU training for those without local GPUs
- **Notebooks**: Skeleton notebook for exploration
- **Scripts**: Example scripts to run your training and evaluation jobs on a slurm cluster or on Modal

Compute Budget:

You have **$500 in Modal credits** for the entire course (all 4 homework). Budget wisely!

Part I: Understanding Your Data (10 points)

Since the goal for generative modeling is to model the distribution of your data, it is important to first understand what does this distribution look like.

The dataset for this course is a custom subset of CelebA [Liu et al., 2015], filtered to have specific properties. Your first task is to discover what those properties are.

Q1. Visual Exploration [5 pts]

(a) [2 pts] Visualize a grid of at least 16 random samples from the training set. Include this grid below.

(c) [3 pts] This dataset was filtered from full CelebA using their 40 binary attributes. Based on your visual exploration, hypothesize which attributes were likely used to create this subset. What filtering criteria were used to create this subset? Why?

Q2. What did you learn? [5 pts]

(a) [3 pts] Imagine you found a pre-trained diffusion model that can reach SOTA performance on full CelebA (200K diverse faces). Now you generated samples and compared them to this filtered subset using KID or FID. Would you expect the FID and KID score to still be SOTA? Why?

(b) [2 pts] Based on your exploration, what kind of data augmentation transformation do you plan to use for your training and why?

Part II: Implement DDPM (25 points)

Now it's time to build your first diffusion model! Take a good look at the starter code and start building! Remember, you are free to add, delete and modify any and all parts of the starter code to fit your preference.

Q3. Building Intuition [optional, 0 pts]

Before diving into the full scale training, it is generally a good idea to build some intuitions of the algorithm with some toy examples as they are easier to debug.

(a) [0 pts] We have prepared a Jupyter notebook *01_1d_playground.ipynb* for you to experiment with 1D toy data. This notebook contains some options of mixture of Gaussians and their visualizations, you can try out your algorithm first in this playground.

(b) [0 pts] Full scale training can be difficult to debug sometimes, and that's why in *train.py* we also provide an argument flag, *--overfit-single-batch*, for you to toggle an experiment where you overfit to a single batch of data for sanity checking. You should be able to iterate your model a lot faster with this experiment than full scale training.

Q4. Implementation [25 pts]

Now implement DDPM for real! The starter code provides the skeleton, and your job is to fill in the core algorithm. Remember: this is your codebase. Feel free to modify any part of the starter code to fit your preferences. Add helper functions, reorganize files, change the config structure... whatever helps. The only requirement is that your final code runs and produces good results.

The starter code provides a detailed guideline on which files are provided and which files are the ones that need your implementation. Go check it out!

(a) [5 pts] Train your DDPM model to convergence. You may use either the provided configs or your own. Report:

1. Model size
2. Batch size
3. Total training iterations
4. Training loss curve
5. Compute cost (GPU hours)

(b) [10 pts] Generate a grid of 16 samples from your trained model. Include this grid below.

(c) [10 pts] Evaluate your model KID with 1k samples and report your KID score (both mean and std). You should be able to obtain KID < 0.005 with single L40S GPU training for a few hours.

(d) [0 pts] Provide a zip file of your code through Gradescope "Homework 1 Code" assignment. Make sure your code is runnable because we may run it to verify your results, and do not include any large files such as model checkpoints or dataset files. If you do not provide a zip file for your code, you will receive 0 point on Part II and Part IV of this homework.

Part III: Help Your "Classmate" Debug (15 points)

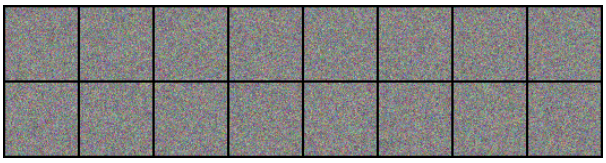
Your classmate Kale is having some trouble getting her model to work. As someone who has successfully trained a DDPM model, you decide to help!

In each of the following scenario, you may be provided a loss curve, a grid of samples or a description of the situation. Try your best to help Kale with her implementation!

Q5. Pseudo Debugging Scenarios [15 pts]

(a) [5 pts] Kale trained a model for 1000 iterations and observed this seemingly nice looking loss curve. However, all her samples are pure noise. What could be the problem here? List at least two potential sources of bugs.

(a) The loss curve she observed



(b) The final samples she obtained

Figure 1: The loss curve and the samples that Kale observed.

(b) [5 pts] Kale observed that after a while her loss would stay at a non-zero level, and it is quite "bumpy" – the loss does not decrease monotonically but instead has small fluctuations. Should she be worried about this? Why does this "bumpiness" exist? Can training for longer still be valuable for improving the model performance in her case?

(c) [5 pts] After the model fully converged, Kale obtained these samples shown below. What is her bug and how to fix it?

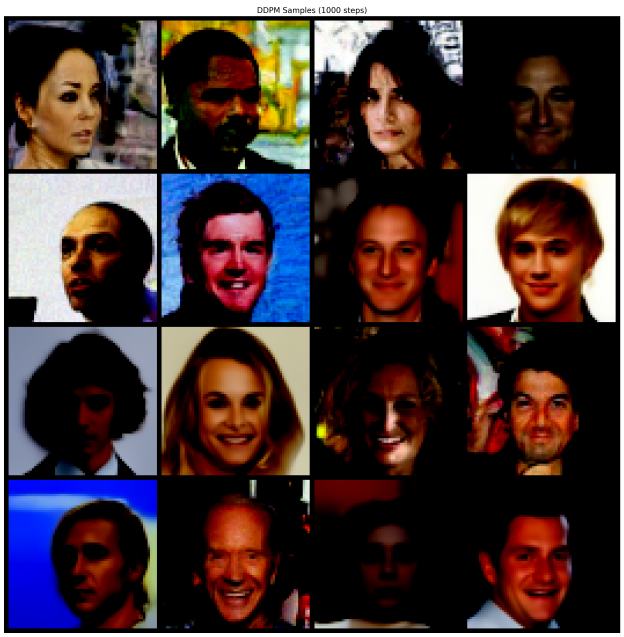


Figure 2: The samples that Kale obtained.

Part IV: Ablation Study (30 points)

In this part of the homework, we will explore different design choices of the DDPM algorithm and how they can affect the final performance.

Q6. Alternative Parametrization [20 pts]

(a) [7 pts] The original DDPM paper parametrized the model to predict the noise ϵ. However, this is not the only option! First derive an alternative parametrization for the same model (i.e. what else can the model predict that can still yield the same mathematical formulation of the algorithm).

(b) [10 pts] Now implement your derived parametrization and train a model with the same configuration with your previous DDPM implementation with the only difference being the model parametrization. Include a grid of 16 samples and report your KID below. (Budget tip: You can compare the two parameterizations using shorter training runs, as long as they provide meaningful comparisons.)

(c) [3 pts] Compare the performance of your new parametrization with that of the original parametrization. Share your findings below.

Q7. Sampling Steps Ablation [10 pts]

(a) [10 pts] DDPM typically uses 1000 timesteps for sampling, which is slow. How about using fewer steps? Report the KID scores for DDPM sampling algorithm with $100, 300, 500, 700, 900$ steps and compare with your original sampling with 1000 steps. Include 1 sample for each number of steps to qualitatively show your comparisons as well.

Part V: Reflection (10 points)

Now that you have had a fully immersive experience with DDPM, let's take a step back and reflect on what we have learned so far.

Q8. Reflection Questions **[10 pts]**

(a) [3 pts] Based on your experience implementing and experimenting with DDPM: What do you think DDPM is bad at? Identify 2-3 limitations you observed or suspect. These could relate to sample quality, training efficiency, sampling speed, controllability, or anything else. Be specific—connect each limitation to something concrete from your experiments or implementation.

(b) [3 pts] For those limitations you identified, how do you think they could be improved? Propose at least 2 concrete ideas. These don't need to be novel—they can be things you've heard about, ideas from papers, or your own speculation. For each idea, briefly explain why you think it might help. (We're not asking you to implement these—just to think critically about the design space.)

(c) [3 pts] What ablations or experiments are you still curious about? List 1-2 things you would explore if you had more time and compute budget. Why do these interest you?

(d) [1 pts] List all the resources that you found helpful for this homework. Include AI tools, open source code, tutorials, papers, classmates that helped you, etc.

That's it! You have completed HW 1! Congrats on your freshly baked diffusion models!

References

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Anton Obukhov, Maximilian Seitzer, Po-Wei Wu, Semen Zhydenko, Jonathan Kyl, and Elvis Yu-Jing Lin. High-fidelity performance metrics for generative models in pytorch, 2020. URL https://github.com/toshas/torch-fidelity. Version: 0.3.0, DOI: 10.5281/zenodo.4957738.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.